Chapter 4.a
MATLAB SPECTRAL ANALYSIS

Using Fourier analysis to determine spectral content of continuous-time signals is straight-forward, and by now, second-nature. Often the results have been attained so many times before that the integrations aren't even required for the task at hand. For sampled signals, the methods of Fourier can still be used (for a derivation see the DFT handout), but some precautions must be observed in order for the results to have any real meaning. If these precautions are not followed, the results obtained may appear to be valid with the user completely unaware that the output is garbage.

Modeling signals with MATLAB requires representing them as sampled signals. This is because a computer cannot generate a time or frequency continuum. Instead, there is a finite displacement between adjacent time or frequency elements. This displacement can be made extremely small, depending on the precision of the computer and its application software, but nonetheless there remains an incremental distance between elements.

Among the considerations which must be pondered in a sampled modeling system are the sampling period or rate, the time and frequency resolutions, and what discrete frequency and inverse discrete frequency transforms will deliver. We begin with the sampling period.

A. SAMPLING PERIOD

For discussion we begin with a simple continuous-time cosine wave of unity amplitude and frequency, i.e., $x(t) = \cos(2\pi t)$. Recall that the Nyquist criterion mandates that we sample at a minimum of twice the highest frequency of the signal. Since the highest (and only) frequency is 1 Hz, the minimum sampling rate f_s is 2 Hz. The sampling period is the inverse of the sampling rate, $T_s = 1/f_s$ so that the greatest sampling period is 0.5 seconds. We are free to set the sampling period at any duration less than this Nyquist period. The top plot of Figure 1 shows x(t) sampled at the Nyquist rate and the middle plot shows the same signal sampled at the higher rate of 5 Hz (oversampled). The bottom plot illustrates undersampling with the sampling rate being 1.25 Hz. From the Nyquist theorem we know that the signal can be reconstructed from the top two plots, but not from the bottom one because of aliasing. This will become apparent in Section C.

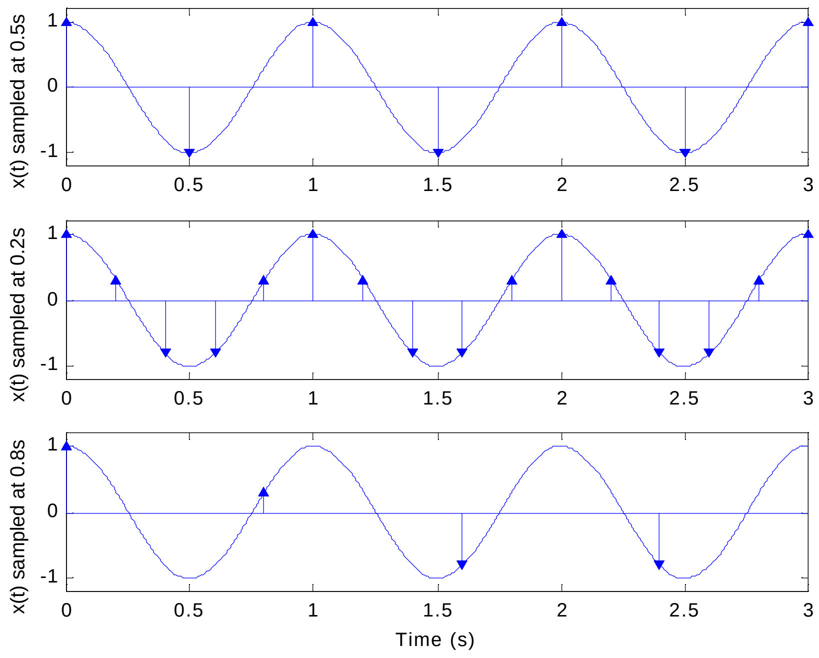

Figure 1. Signal sampled at three different rates.

If x(t) is comprised by more than one frequency, e.g., 1 and 10 Hz resulting in $x(t) = \cos(2\pi t) + 0.2\cos(2\pi(10)t)$, then the signal must be sampled at a rate at least as great as 20 Hz.

It is obvious from Figure 1 that the higher the sampling rate the more samples, N, we generate. From the top plot the sampled values are $x_s = \{1, –1, 1, –1, 1, –1, 1\}$, (N = 7), from the middle plot {1, 0.3, –0.8, –0.8, 0.3, 1, 0.3, –0.8, –0.8, 0.3, 1, 0.3, –0.8, –0.8, 0.3, 1}, (N = 16), while the sequence from the bottom plot is {1, 0.3, –0.8, –0.8}, (N = 4). More samples require more overhead time to gather and store, and more storage space on the hard drive. Furthermore, more samples require more time in the CPU to process the data with filtering, frequency analysis, etc. Since the requirements of Nyquist tell us that the signal can be reconstructed if we sample at least twice the highest frequency, which generates the fewest permissible samples, it would seem that this is the preferred sampling rate. Unfortunately, that may not be the case as shown in the next section.

B. TIME AND FREQUENCY RESOLUTION

It is obvious that the shorter the sampling period (or higher the sampling rate), the better time resolution Δt of the model. To illustrate why this might be important suppose the waveform of a power generator is being monitored to ensure output quality. Its normal output is a 1-Hz signal with a constant 1 volt peak amplitude. Now suppose a bearing is failing which causes a voltage spike 0.4 seconds in duration in the generator at every revolution at the same rotational location. This induces a spike at a constant point in its output sine wave. If we are sampling at the Nyquist rate and the spike occurs during a sample time, the spike will be detected and the necessary repairs effected before further damage occurs. If instead, the spike occurs in between samples as shown in Figure 2, it will not be detected from the sampled signal.

It is apparent from this example that the time resolution is equal to the sample period, i.e., $\Delta t = T_s$. Therefore if a waveform's duration is less than the sample period it might not be detected from the sampling. This illustrates that although sampling at the Nyquist rate gives us the fewest samples and therefore the easiest processing, the tradeoff is that required time resolution may be forfeited.

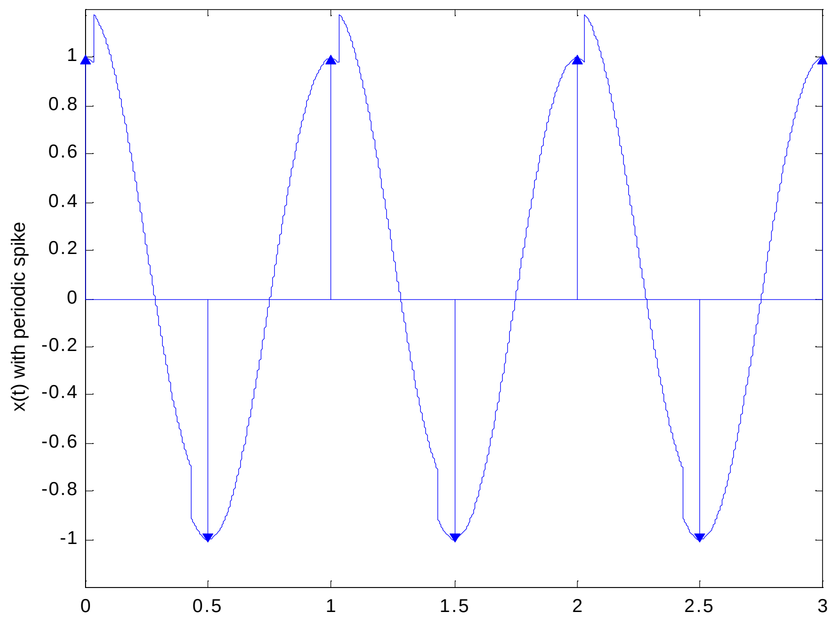

Figure 2. Voltage spike occurring between samples.

We now turn our attention to frequency resolution. It is seen that even though the repetition rate of the spike in this example is less than that of the cosine wave (contributing a lower frequency), the frequencies which make up the spike are much higher than those of the cosine wave. However, it is seen that if the spike is not detected in the time domain it will not be discerned in the frequency domain either. It would seem obvious that the time resolution and frequency resolution are linked, and indeed they are.

We know that time and frequency are inversely related so our first guess at the frequency resolution might be the inverse of the time resolution. And since the time resolution is equal to the sampling period T_s, its inverse is the sampling rate f_s. A little thought reveals that this is not the frequency resolution but the highest frequency that can be manifested with the model, i.e.,

$$f_{\max} = f_s. \tag{1}$$

The frequency resolution is then f_{max} divided by the number of frequency cells or

$$\Delta f = \frac{I_s}{N} = \frac{1}{NT_s} = \frac{1}{N\Delta t}\ Hz. \qquad (2)$$

(Recall that the number of frequency cells is identical to the number of samples.) From this equation we see that frequency resolution is affected by two quantities, the sampling rate or period, and the total number of samples. Although N is dependent upon Δt, it also depends on the length of the sampling interval (which is 3 seconds in the above example).

Although the frequency domain of the sampled signal indicates a maximum frequency of f_s, this fact contributes a source of confusion because the highest frequency which can exist from the original signal is half that or $f_s/2$. This apparent discrepancy can lead the newcomer to discrete frequency analysis to make some critical errors in evaluating the DFT. With this caution we now discuss spectral analysis using MATLAB.

C. DISCRETE FREQUENCY AND INVERSE FREQUENCY TRANSFORMS

As discussed in the DFT handout there are two methods to find the Fourier transform of a sampled signal. The Discrete Fourier Transform (DFT) is derived in the handout and the other method is the Fast Fourier Transform (FFT) you encountered in a previous course. The two methods deliver the same results, but the FFT is much faster and requires fewer computer computational resources. They are both obtained in MATLAB using the same command, fft. The difference is if the number of sample points is equal to a power of 2 the FFT is performed whereas the DFT is used otherwise. For a small number of samples N, it is irrelevant which method is used. However, for large N the difference in computation time can be significant.

If we find the FFT of the three sampled signals shown in Figure 1 we obtain generally meaningless results. This is because there are too few sample points for the FFT to gravitate toward the frequency content (with the possible exception of the oversampled case with 16 points, try it and see!). The solution is to allow the sampling interval I to increase from 3 seconds to, say, 30 seconds. With I = 30 seconds for the three signals of Figure 1 the FFTs were computed using MATLAB and the magnitude of the results shown in Figure 3.

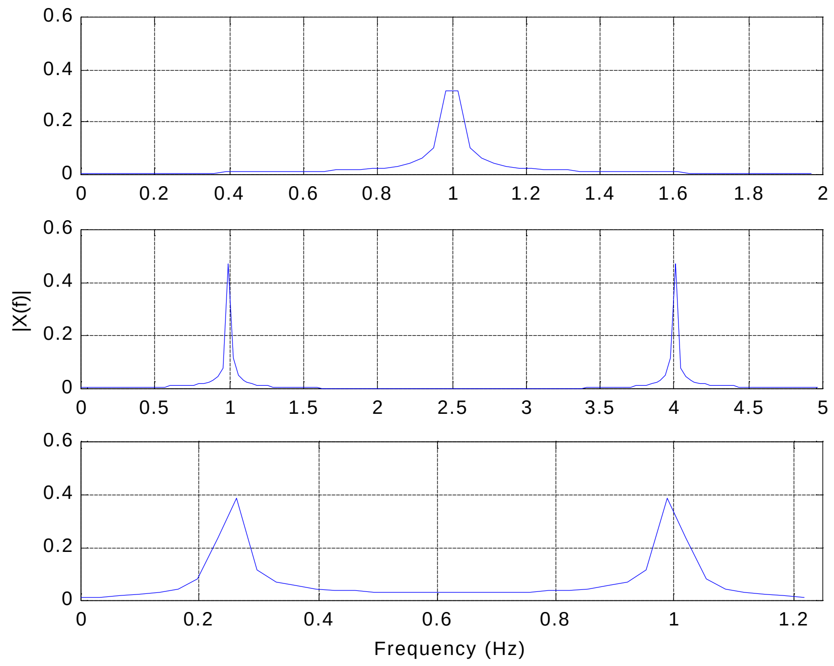

Figure 3. FFT magnitudes for three signals from Figure 1. The top plot is from the Nyquist rate sampled signal, the middle is oversampled and the bottom is undersampled.

Some discussion of this figure is warranted. Notice first that the frequency span of each plot extends to f_{max}, the sampling frequency. However, since the maximum input from the signal is $f_{max}/2$, only half the plot is relevant; the second half is only a mirror image of the left half (except for the undersampled case which we discuss in a moment). In the top plot the sampling frequency is 2 Hz, so that the only unique input is at 1 Hz and below. So we see the 1 Hz signal plotted at the limit of the relevant plot. The mirror image continues from that point on, but it is repeat information. In the middle plot, f_{max} is 5 so the mirror point is at 2.5 Hz. Notice the signal magnitude plotted at 1 Hz where it belongs but its image is at 4 Hz. This pulse does not exist; there is no 4-Hz component. The bottom plot is undersampled which is forbidden under Nyquist criteria. We see why. The 1-Hz pulse is present, but its image has been folded back as an alias frequency at 0.25 Hz.

If you weren't careful you would interpret the alias frequency as the true output and the true pulse as the image. So, it is apparent that we never want to undersample, and for an unmistakable output, we prefer to oversample. For the remaining examples we will only discuss the oversampled case.

There are two solutions to preventing the image frequency from confusing the user. Some people plot only the first half of the points. This is shown in Figure 4 for the oversampled case.

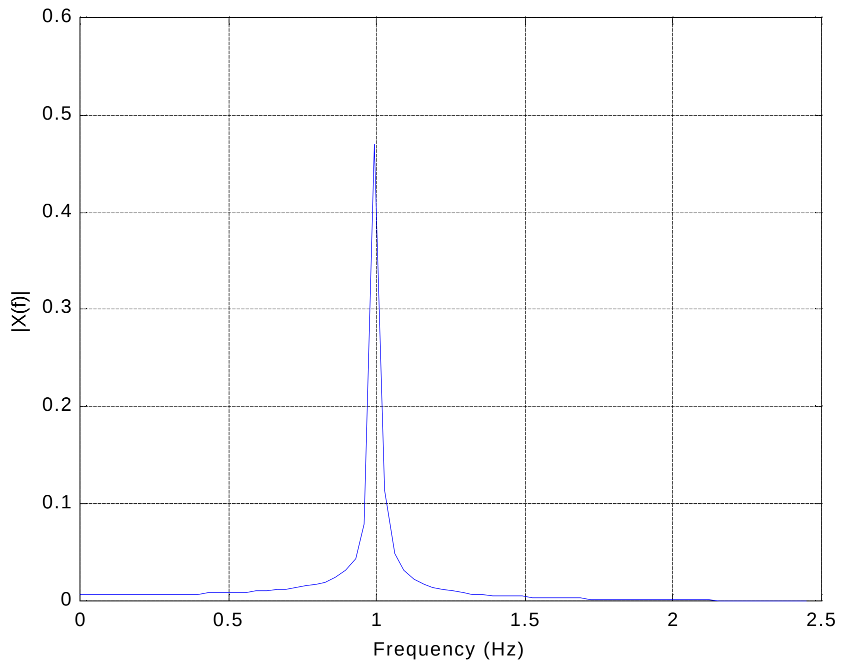

Figure 4. One Hz signal with only half of FFT plotted.

Another method to prevent confusion is to make the FFT output look like the theoretical output. Recall from theory that the Fourier transform of a sinusoid results in a pulse at both positive and negative frequencies, in our case ± 1 Hz. This can be displayed by moving the image frequency (in the oversampled case at 4 Hz) to –1 Hz where it belongs. This is accomplished with the MATLAB command fftshift. Applying fftshift to the middle plot of Figure 3 results in Figure 5. What fftshift does is take everything to the right of $f_s/2$, reverse it, and shift it to the left of DC.

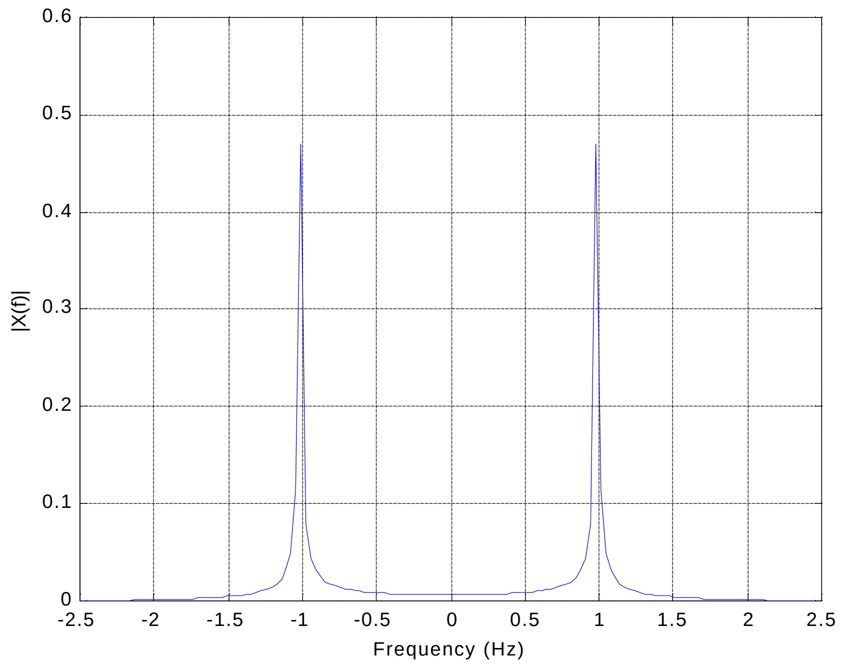

Figure 5. FFT with positive and negative frequencies.

A note about using the fft function in MATLAB. The program uses the same algorithm for computing both the fft and the inverse fft. For this reason it does not properly scale the output. You will recall from theory that the fft summation must be divided by N, the number of sample points. MATLAB does not include this division so you must provide it yourself. Similarly, when executing the inverse fft, ifft, theory requires that you multiply by N. Again, MATLAB does not perform this multiplication, so you must provide it in your code.

The FFTs shown above are only the magnitude of X(f). The fft (and Fourier transform) is complex for real time domain signals. We have not plotted the phase, only the magnitude using the abs function.

Now let's take what we've learned and see how we can mangle the inverse FFT. If the inverse operation works properly then we should be able to recover whatever time domain signal we began the transformations with. Let's see if we can.

For comparison first we plot the oversampled signal for which the FFTs of Figures 4 and 5 and the middle plot of Figure 3 were derived. This is shown in Figure 6. This is the signal that we wish to get back from the inverse FFT.

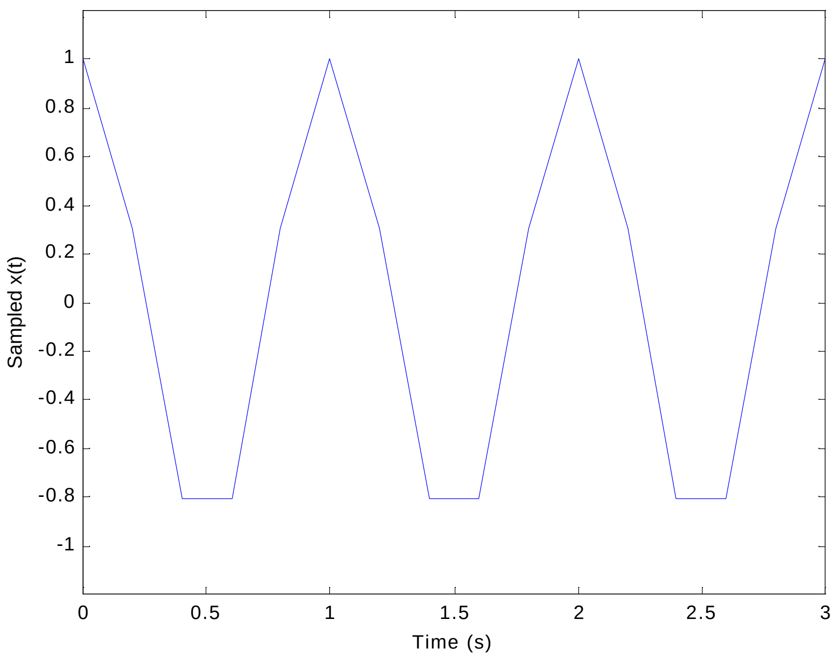

Figure 6. Sampled one Hertz signal.

Beginning with the frequency domain signal shown in Figure 5 we find its inverse FFT. First we find the ifft of the plot just as it is pictured, i.e., the magnitude of X(f). The result is shown in the top plot of Figure 7. As this does not look like Figure 6 we decide that we shouldn't use the magnitude of the FFT in the calculations but the entire complex FFT. The ifft of that is shown in the bottom part of Figure 7. It likewise does not look like Figure 6.

So, we decide that the fftshift might have messed us up. We return to the X(f) shown in the middle plot of Figure 3. Using this signal to perform the ifft on we get the time domain signal shown in Figure 8. We see that this is the proper waveform.

For the ifft we have learned that we cannot use the fftshifted signal. If the signal is in this form we must again apply fftshift to get it back to the original form. Also, we know that we must use the complex version of the FFT not just its magnitude. Otherwise we will not get the proper output signal.

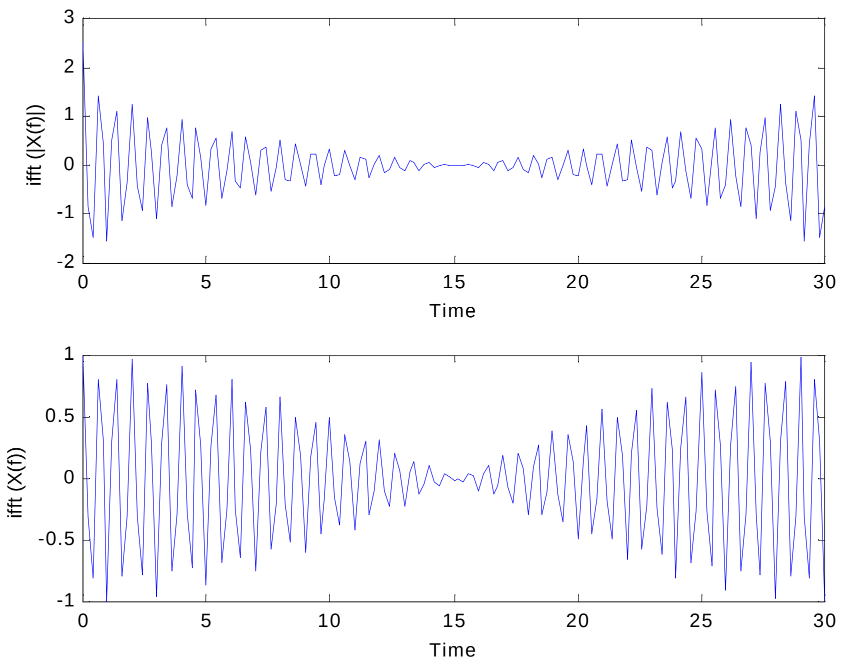

Figure 7. Inverse FFT of |X(f)| and X(f) from Figure 5.

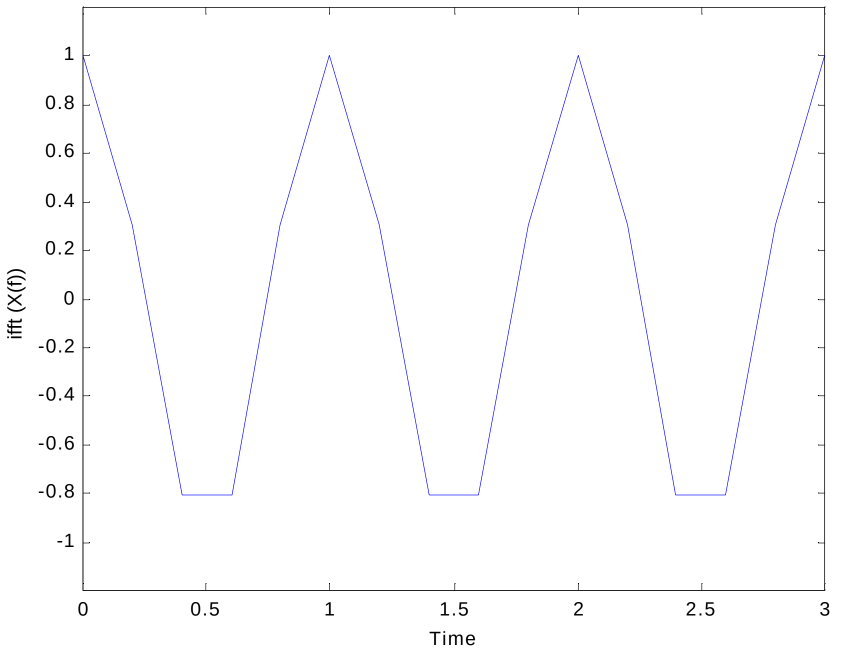

Figure 8. Inverse FFT of middle plot of Figure 3.

D. DESIGNING LOW-PASS FILTERS

Upon finding the output from the system shown in Figure 8, we almost feel betrayed. We observed the Nyquist criteria, but the signal of Figure 8 does not look like our original cosine signal. However, the Nyquist criteria states that we can recover the original signal. The recovery process is through filtering.

It is obvious from the figure that there are higher frequencies than contained in the original 1-Hz signal. We therefore infer that we need to low-pass filter the signal in order to recover the original signal. We wish to use MATLAB and know that we have several choices of filter types, and we wish to use a butterworth.

Consulting the help function for *butter*, it asks us to supply two pieces of information. The first is the order of the filter. The higher the order, the steeper the transition region of the filter, but the more unstable it is. A good compromise is a 5^{th} order filter. The second piece of information is something called "Wn" and we are told that it is defined between 0 and 1. We are further told that a 1 value corresponds to half the sampling frequency. This W_n represents the cutoff frequency of the filter.

Thinking back to the above discussions we know that $f_s/2$ is the maximum frequency that can be represented in the sampled signal. There $W_n = 1$ corresponds to the maximum frequency. If we used this value no filtering would occur as all the frequencies would be allowed to pass. Conversely, if we used $W_n = 0$ no frequencies would pass as the cutoff frequency would be below all frequencies of the signal.

For our particular example, $f_s/2$ is 2.5 Hz, which would be represented by $W_n = 1$. Our signal is at 1 Hz. To find W_n for 1 Hz we divide by $f_s/2$ to get $1/2.5 = 0.4$. We may want to pad this a bit so we might set $W_n = 0.42$. We see from the help function that we will get b and a values from butter using [b,a] = butter(5,0.42).

Now, from help filter, we find that the filter function requires these b and a values and whatever signal we wish to filter, x. The output is the filtered value as shown in Figure 9.

We got the results shown in this figure by using the butterworth filter designed in the easy steps above. One additional step was added to get this

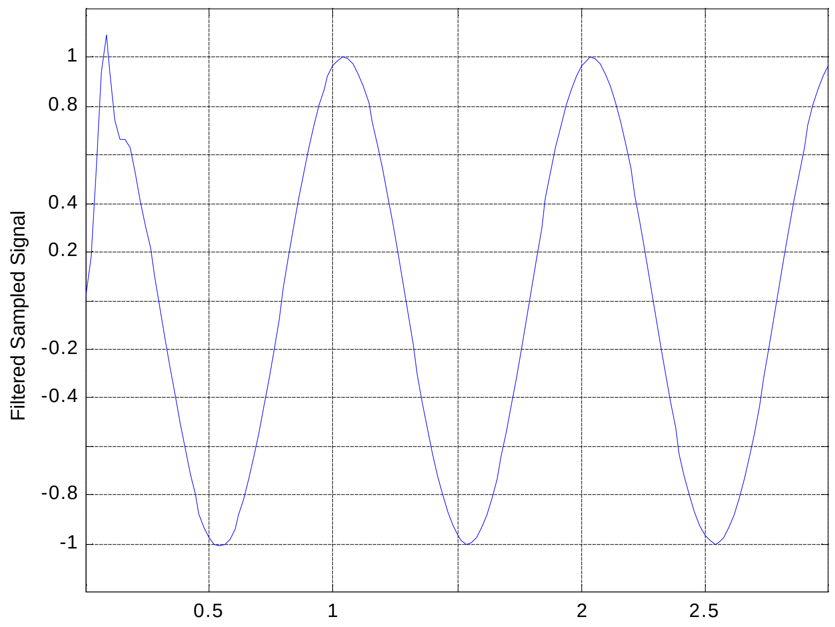

Figure 9. Signal of Figure 8 filtered with a low-pass butterworth at cutoff frequency 1.05 Hz.

waveform. Since we are simulating a real analog system, one that is not sampled, there will be an infinite number of values to fill in between the sampled values. To simulate this we add additional points to the sampled waveform (we do not change the shape of the sampled waveform, just fill in between points). We do this with the MATLAB command interp (interpolation). In this example we interpolated by increasing the number of points ten fold.

Also notice in the figure that there are transients at startup, just as in a real system, and that the waveform has been delayed slightly due to the filtering (just as in a real system).

E. MATLAB FILES

The following MATLAB files were used to create the examples in this Chapter.

```
% nyq.m
T = 30; % Sampling interval
dt1 = .5; % Sampling period
t1 = 0:dt1:T; % Sample time vecto
x1 = cos(2*pi*t1); % Sampled signal
arrowtop(t1,x1,'filled') % Plot arrows at sample points, my own program-use stem
hold on
t = 0:.0001:3; % Time vector for "continuous" cas
x = cos(2*pi*t);% Continuous signal
subplot(3,1,1)
plot(t,x)
axis([0 3 -1.2 1.2])
%xlabel('Time (s)')
ylabel('x(t) sampled at 0.5s')
subplot(3,1,2)
dt2 = .2;
t2 = 0:dt2:T;
x2 = cos(2*pi*t2);
arrowtop(t2,x2,'filled')
hold on
t = 0:.0001:3;
x = cos(2*pi*t);
plot(t,x)
axis([0 3 -1.2 1.2])
%xlabel('Time (s)')
ylabel('x(t) sampled at 0.2s')
```

```
subplot(3,1,3)
dt3 = .8;
t3 = 0:dt3:T;
x3 = cos(2*pi*t3);
arrowtop(t3,x3,'filled')
hold on
t = 0:.0001:3;
x = cos(2*pi*t);
plot(t,x)
axis([0 3 -1.2 1.2])
xlabel('Time (s)')
ylabel('x(t) sampled at 0.8s')

%This m file is used to generate Figure 3.
%Note that the m file nyq.m must be run first.
fmax = 1/dt1;
N = length(t1);
df = fmax/(N);
f = 0:N-1;
f = f*df;
y = fft(x1)/N/2;
subplot(3,1,1)
plot(f,abs(y)),grid
axis([ 0 fmax 0 .6])
fmax = 1/dt2;
N = length(t2);
df = fmax/(N);
f = 0:N-1;
f = f*df;
y = fft(x2)/(N);
subplot(3,1,2)
```

```
plot(f,abs(y)),grid
axis([ 0 fmax 0 .6])
ylabel('|X(f)|')
fmax = 1/dt3;
N = length(t3);
df = fmax/(N);
f = 0:N-1;
f = f*df;
y = fft(x3)/(N);
subplot(3,1,3)
plot(f,abs(y)),grid
axis([ 0 fmax 0 .6])
xlabel('Frequency (Hz)')

%m file for Figure 4
fmax = 1/dt2;
N = length(t2);
df = fmax/(N);
f = 0:N-1;
f = f*df;
y = fft(x2)/(N);
plot(f(1:N/2),abs(y(1:N/2))),grid
axis([ 0 fmax/2 0 .6])
ylabel('|X(f)|')
xlabel('Frequency (Hz)')

% m file for Figures 5, 7, and 8
fmax = 1/dt2;
N = length(t2);
df = fmax/(N);
f = 0:N-1;
```

```
f = f*df;
y = fft(x2)/(N);
y = fftshift(y);
f = f - fmax/2;
plot(f,abs(y)),grid
axis([ -fmax/2 fmax/2 0 .6])
ylabel('|X(f)|')
xlabel('Frequency (Hz)')
pause
subplot(2,1,1)
x = N*ifft(abs(y));
plot (t2,x)
xlabel('Time')
ylabel('ifft (|X(f)|)')
subplot(2,1,2)
x = N*ifft(y);
plot (t2,x)
xlabel('Time')
ylabel('ifft (X(f))')
pause
close
fmax = 1/dt2;
N = length(t2);
df = fmax/(N);
f = 0:N-1;
f = f*df;
y = fft(x2)/(N);
x = ifft(y) * N;
plot(t2,x)
axis([0 3 -1.2 1.2])
xlabel('Time')
```

```
ylabel('ifft (X(f))')

%  Figures 6 and 9
T = 30;
dt2 = .2;
t2 = 0:dt2:T;
x2 = cos(2*pi*t2);
plot(t2,x2)
axis([0 3 -1.2 1.2])
xlabel('Time (s)')
ylabel('Sampled x(t)')
pause
[b,a] = butter(5,0.42);
Q = 10;
x = interp(x2,Q);
xfilter = filter(b,a,x);
L = length(xfilter);
dt = dt2/Q;
t = 0:dt:T;
plot(t(1:500),xfilter(1:500)),grid
axis([0 3 -1.2 1.2])
xlabel('Time (s)')
ylabel('Filtered Sampled Signal')
```